MLGmpIDL: OCaml interface for GMP library

December 19, 2006

All files distributed in the MLGmpIDL interface are distributed under LGPL license.
Copyright (C) Bertrand Jeannet 2005-2006 for the MLGmpIDL interface.

Introduction

This package is an OCaml interface for the GMP interface, which is decomposed into 15 submodules, corresponding to C modules:

Mpz	:	integers, with side-effect semantics (as in C library)
Mpq	:	rationals, with side-effect semantics (as in C library)
Mpzf	:	integers, with functional semantics
Mpqf	:	rationals, with functional semantics
Gmp_random	:	random number functions

Requirements

- GMP library (tested with version 4.0 and up)
- OCaml 3.0 or up (tested with 3.09)
- Camlidl (tested with 1.05)

Installation

Library Set the file ../Makefile.config to your own setting. You might also have to modify the Makefile for executables

type 'make', possibly 'make debug', and then 'make install'

The OCaml part of the library is named gmp.cma (.cmxa, .a) The C part of the library is named libgmp_caml.a (libgmp_caml_debug.a)

'make install' installs not only .mli, .cmi, but also .idl files.

Documentation The documentation (currently very sketchy) is generated with ocamldoc.

'make mlapronidl.dvi'

'make html' (put the HTML files in the html subdirectoy)

Miscellaneous 'make clean' and 'make distclean' have the usual behaviour.

Contents

Chapter 1

Module Mpz : GMP multi-precision integers

`type t`

GMP multi-precision integers

The following operations are mapped as much as possible to their C counterpart. In case of imperative functions (like `set`, `add`, ...) the first parameter of type `t` is an out-parameter and holds the result when the function returns. For instance, `add x y z` adds the values of `y` and `z` and stores the result in `x`.

These functions are as efficient as their C counterpart: they do not imply additional memory allocation, unlike the corresponding functions in the module `Mpzf`[4].

1.1 Pretty printing

```
val print : Format.formatter -> t -> unit
```

1.2 Initialization Functions

```
val init : unit -> t
val init2 : int -> t
val realloc2 : t -> int -> unit
```

1.3 Assignement Functions

The first parameter holds the result.

```
val set : t -> t -> unit
val set_si : t -> int -> unit
val set_d : t -> float -> unit
val set_str : t -> string -> int -> bool
val swap : t -> t -> unit
```

1.4 Combined Initialization and Assignment Functions

```
val init_set : t -> t
```

```
val init_set_si : int -> t
val init_set_d : float -> t
val init_set_str : string -> int -> t
```

1.5 Conversion Functions

```
val get_si : t -> nativeint
val get_d : t -> float
val get_d_2exp : t -> float * int
val get_str : int -> t -> string
```

1.6 User Conversions

These functions are additions to or renaming of functions offered by the C library.

```
val to_string : t -> string
val to_float : t -> float
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
```

1.7 Arithmetic Functions

The first parameter holds the result.

```
val add : t -> t -> t -> unit
val add_ui : t -> t -> int -> unit
val sub : t -> t -> t -> unit
val sub_ui : t -> t -> int -> unit
val ui_sub : t -> int -> t -> unit
val mul : t -> t -> t -> unit
val mul_si : t -> t -> int -> unit
val addmul : t -> t -> t -> unit
val addmul_ui : t -> t -> int -> unit
val submul : t -> t -> t -> unit
val submul_ui : t -> t -> int -> unit
val mul_2exp : t -> t -> int -> unit
val neg : t -> t -> unit
val abs : t -> t -> unit
```

1.8 Division Functions

`c` stands for ceiling, `f` for floor, and `t` for truncate (rounds toward 0).

1.8.1 Ceiling division

```
val cdiv_q : t -> t -> t -> unit
```

The first parameter holds the quotient.

```
val cdiv_r : t -> t -> t -> unit
```
The first parameter holds the remainder.

```
val cdiv_qr : t -> t -> t -> t -> unit
```
The two first parameters hold resp. the quotient and the remainder).

```
val cdiv_q_ui : t -> t -> int -> int
```
The first parameter holds the quotient.

```
val cdiv_r_ui : t -> t -> int -> int
```
The first parameter holds the remainder.

```
val cdiv_qr_ui : t -> t -> t -> int -> int
```
The two first parameters hold resp. the quotient and the remainder).

```
val cdiv_ui : t -> int -> int
val cdiv_q_2exp : t -> t -> int -> unit
```
The first parameter holds the quotient.

```
val cdiv_r_2exp : t -> t -> int -> unit
```
The first parameter holds the remainder.

1.8.2 Floor division

```
val fdiv_q : t -> t -> t -> unit
val fdiv_r : t -> t -> t -> unit
val fdiv_qr : t -> t -> t -> t -> unit
val fdiv_q_ui : t -> t -> int -> int
val fdiv_r_ui : t -> t -> int -> int
val fdiv_qr_ui : t -> t -> t -> int -> int
val fdiv_ui : t -> int -> int
val fdiv_q_2exp : t -> t -> int -> unit
val fdiv_r_2exp : t -> t -> int -> unit
```

1.8.3 Truncate division

```
val tdiv_q : t -> t -> t -> unit
val tdiv_r : t -> t -> t -> unit
val tdiv_qr : t -> t -> t -> t -> unit
val tdiv_q_ui : t -> t -> int -> int
val tdiv_r_ui : t -> t -> int -> int
val tdiv_qr_ui : t -> t -> t -> int -> int
val tdiv_ui : t -> int -> int
val tdiv_q_2exp : t -> t -> int -> unit
val tdiv_r_2exp : t -> t -> int -> unit
```

1.8.4 Other division-related functions

```
val gmod : t -> t -> t -> unit
val gmod_ui : t -> t -> int -> int
val divexact : t -> t -> t -> unit
val divexact_ui : t -> t -> int -> unit
val divisible_p : t -> t -> bool
val divisible_ui_p : t -> int -> bool
val divisible_2exp_p : t -> int -> bool
val congruent_p : t -> t -> t -> bool
val congruent_ui_p : t -> int -> int -> bool
val congruent_2exp_p : t -> t -> int -> bool
```

1.9 Exponentiation Functions

```
val powm : t -> t -> t -> t -> unit
val powm_ui : t -> t -> int -> t -> unit
val pow_ui : t -> t -> int -> unit
val ui_pow_ui : t -> int -> int -> unit
```

1.10 Root Extraction Functions

```
val root : t -> t -> int -> bool
val sqrt : t -> t -> unit
val sqrtrem : t -> t -> t -> unit
val perfect_power_p : t -> bool
val perfect_square_p : t -> bool
```

1.11 Number Theoretic Functions

```
val probab_prime_p : t -> int -> int
val nextprime : t -> t -> unit
val gcd : t -> t -> t -> unit
val gcd_ui : t option -> t -> int -> int
val gcdext : t -> t -> t -> t -> t -> unit
val lcm : t -> t -> t -> unit
val lcm_ui : t -> t -> int -> unit
val invert : t -> t -> t -> bool
val jacobi : t -> t -> int
val legendre : t -> t -> int
val kronecker : t -> t -> int
val kronecker_si : t -> int -> int
val si_kronecker : int -> t -> int
val remove : t -> t -> t -> int
val fac_ui : t -> int -> unit
val bin_ui : t -> t -> int -> unit
```

```
val bin_uiui : t -> int -> int -> unit
val fib_ui : t -> int -> unit
val fib2_ui : t -> t -> int -> unit
val lucnum_ui : t -> int -> unit
val lucnum2_ui : t -> t -> int -> unit
```

1.12 Comparison Functions

```
val cmp : t -> t -> int
val cmp_d : t -> float -> int
val cmp_si : t -> int -> int
val cmpabs : t -> t -> int
val cmpabs_d : t -> float -> int
val cmpabs_ui : t -> int -> int
val sgn : t -> int
```

1.13 Logical and Bit Manipulation Functions

```
val gand : t -> t -> t -> unit
val ior : t -> t -> t -> unit
val xor : t -> t -> t -> unit
val com : t -> t -> unit
val popcount : t -> int
val hamdist : t -> t -> int
val scan0 : t -> int -> int
val scan1 : t -> int -> int
val setbit : t -> int -> unit
val clrbit : t -> int -> unit
val tstbit : t -> int -> bool
```

1.14 Input and Output Functions: not interfaced

1.15 Random Number Functions: see Gmp_random[3] module

1.16 Integer Import and Export Functions

```
val import :
  t ->
  (int, Bigarray.int32_elt, Bigarray.c_layout) Bigarray.Array1.t ->
  int -> int -> unit
val export :
  t ->
  int ->
  int -> int -> (int, Bigarray.int32_elt, Bigarray.c_layout) Bigarray.Array1.t
```

1.17 Miscellaneous Functions

```
val fits_ulong_p : t -> bool
val fits_slong_p : t -> bool
val fits_uint_p : t -> bool
val fits_sint_p : t -> bool
val fits_ushort_p : t -> bool
val fits_sshort_p : t -> bool
val odd_p : t -> bool
val even_p : t -> bool
val size : t -> int
val sizeinbase : t -> int -> int
```

Chapter 2

Module Mpq : GMP multiprecision rationals

`type t`

GMP multiprecision rationals

The following operations are mapped as much as possible to their C counterpart. In case of imperative functions (like `set`, `add`, ...) the first parameter of type `t` is an out-parameter and holds the result when the function returns. For instance, `add x y z` adds the values of `y` and `z` and stores the result in `x`.

These functions are as efficient as their C counterpart: they do not imply additional memory allocation, unlike the corresponding functions in the module `Mpqf`[5].

`val canonicalize : t -> unit`

2.1 Pretty printing

`val print : Format.formatter -> t -> unit`

2.2 Initialization and Assignment Functions

`val init : unit -> t`

`val set : t -> t -> unit`

`val set_z : t -> Mpz.t -> unit`

`val set_si : t -> int -> int -> unit`

`val set_str : t -> string -> int -> bool`

`val swap : t -> t -> unit`

2.3 Additional Initialization and Assignements functions

These functions are additions to or renaming of functions offered by the C library.

`val init_set : t -> t`

`val init_set_z : Mpz.t -> t`

`val init_set_si : int -> int -> t`

`val init_set_str : string -> int -> t`

`val init_set_d : float -> t`

2.4 Conversion Functions

```
val get_d : t -> float
val set_d : t -> float -> unit
val get_str : int -> t -> string
```

2.5 User Conversions

These functionss are additions to or renaming of functions offeered by the C library.

```
val to_string : t -> string
val to_float : t -> float
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
val of_frac : int -> int -> t
val of_mpz : Mpz.t -> t
val of_mpz2 : Mpz.t -> Mpz.t -> t
```

2.6 Arithmetic Functions

```
val add : t -> t -> t -> unit
val sub : t -> t -> t -> unit
val mul : t -> t -> t -> unit
val mul_2exp : t -> t -> int -> unit
val div : t -> t -> t -> unit
val div_2exp : t -> t -> int -> unit
val neg : t -> t -> unit
val abs : t -> t -> unit
val inv : t -> t -> unit
```

2.7 Comparison Functions

```
val cmp : t -> t -> int
val cmp_si : t -> int -> int -> int
val sgn : t -> int
val equal : t -> t -> bool
```

2.8 Applying Integer Functions to Rationals

```
val get_num : Mpz.t -> t -> unit
val get_den : Mpz.t -> t -> unit
val set_num : t -> Mpz.t -> unit
val set_den : t -> Mpz.t -> unit
```

2.9 Input and Output Functions: not interfaced

Chapter 3

Module Gmp_random : GMP random generation functions

```
type state
```
GMP random generation functions

3.1 Random State Initialization

```
val init_default : unit -> state
val init_lc_2exp : Mpz.t -> int -> int -> state
val init_lc_2exp_size : int -> state
```

3.2 Random State Seeding

```
val seed : state -> Mpz.t -> unit
val seed_ui : state -> int -> unit
```

3.3 Random Number Functions

```
val urandomb : Mpz.t -> state -> int -> unit
val urandomm : Mpz.t -> state -> Mpz.t -> unit
val rrandomb : Mpz.t -> state -> int -> unit
```

Chapter 4

Module `Mpzf` : GMP multi-precision integers, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in `Mpz`[1]. These functions are less efficients, due to the additional memory allocation neded for the result.

`type t`

multi-precision integer

`val to_mpz : t -> Mpz.t`

`val of_mpz : Mpz.t -> t`

Conversion from and to Mpz.t.

There is no sharing between the argument and the result.

4.1 Constructors

```
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
```

4.2 Conversions and Printing

```
val to_string : t -> string
val to_float : t -> float
val print : Format.formatter -> t -> unit
```

4.3 Arithmetic Functions

```
val add : t -> t -> t
val add_int : t -> int -> t
val sub : t -> t -> t
val sub_int : t -> int -> t
val mul : t -> t -> t
val mul_int : t -> int -> t
```

```
val cdiv_q : t -> t -> t
val cdiv_r : t -> t -> t
val cdiv_qr : t -> t -> t * t
val fdiv_q : t -> t -> t
val fdiv_r : t -> t -> t
val fdiv_qr : t -> t -> t * t
val tdiv_q : t -> t -> t
val tdiv_r : t -> t -> t
val tdiv_qr : t -> t -> t * t
val divexact : t -> t -> t
val gmod : t -> t -> t
val gcd : t -> t -> t
val lcm : t -> t -> t
val neg : t -> t
val abs : t -> t
```

4.4 Comparison Functions

```
val cmp : t -> t -> int
val cmp_int : t -> int -> int
val sgn : t -> int
```

Chapter 5

Module Mpqf : GMP multi-precision rationals, functional version

Functions in this module has a functional semantics, unlike the corresponding functions in Mpq[2]. These functions are less efficients, due to the additional memory allocation neded for the result.

```
type t
```

multi-precision rationals

```
val to_mpq : t -> Mpq.t
val of_mpq : Mpq.t -> t
val of_mpz : Mpz.t -> t
val of_mpz2 : Mpz.t -> Mpz.t -> t
```

Conversion from and to Mpz.t and Mpz.t.

There is no sharing between the argument and the result.

5.1 Constructors

```
val of_string : string -> t
val of_float : float -> t
val of_int : int -> t
val of_frac : int -> int -> t
val of_mpzf : Mpzf.t -> t
val of_mpzf2 : Mpzf.t -> Mpzf.t -> t
```

5.2 Conversions and Printing

```
val to_string : t -> string
val to_float : t -> float
val print : Format.formatter -> t -> unit
```

5.3 Arithmetic Functions

```
val add : t -> t -> t
```

```
val sub : t -> t -> t
val mul : t -> t -> t
val div : t -> t -> t
val neg : t -> t
val abs : t -> t
val inv : t -> t
val equal : t -> t -> bool
```

5.4 Comparison Functions

```
val cmp : t -> t -> int
val cmp_int : t -> int -> int
val cmp_frac : t -> int -> int -> int
val sgn : t -> int
```

5.5 Extraction Functions

```
val get_num : t -> Mpzf.t
val get_den : t -> Mpzf.t
```

Index